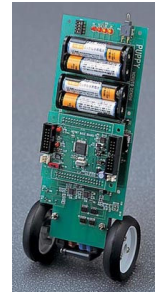
XMOSチップによる自走式倒立制御実験装置の開発

北海道職業能力開発大学校
制御技術科　　中原　博史

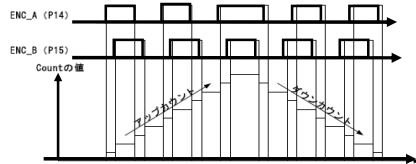
（株）北斗電子　　中野　隆司

自走式倒立制御実験装置

PUPPY
- 倒立制御学習キット
- （株）北斗電子
- http://www.hokutodenshi.co.jp/

これをXMOSチップで制御

PUPPY本体の構成-1/4

- モータ、ギアボックス
- ロータリエンコーダ

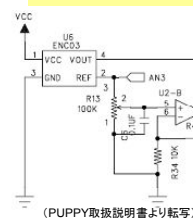
ENC_A (P14)
ENC_B (P15)
Countの値
アップカウント
ダウンカウント

- 前進、後進
- 回転角（ϕ）

（PUPPY取扱説明書より転写）

PUPPY本体の構成-2/4

- 傾斜角・角速度検出
- 圧電振動レイトジャイロ

（PUPPY取扱説明書より転写）

- 角速度、d θ /dt
- 傾斜角 θ は、角速度を積分

PUPPY本体の構成-3/4

モータトルク（ τ ）の検出

- トルクと電流は比例
- 電流は、抵抗の電圧に比例
- 電圧をADコンバータで観測、間接的にトルクを検出する。

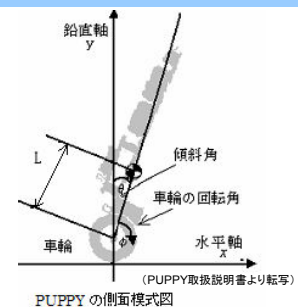
- 倒立状態を維持するトルク（ τ ）になるように電流をPWM方式で制御する。

PUPPY本体の構成-4/4

倒立状態を維持するモータトルク（ τ ）を求め、その τ を実現するように電流を制御する。

鉛直軸 y
L
傾斜角
車輪の回転角
車輪
水平軸 x

（PUPPY取扱説明書より転写）

PUPPY の側面模式図

PUPPYのパラメータ

車輪半径	r	0.029 $[m]$
車輪(2 つ分)と車軸の 質量	m	0.06 $[kg]$
本体の 質量(乾電池含む)	M	0.27 $[kg]$
車輪と車軸の慣性モーメント	J	0.00115 $[kg \cdot m^2]$
本体の 慣性モーメント	I	0.0036973 $[kg \cdot m^2]$
車軸から本体重心までの距離	L	0.105 $[m]$
車体の粘性摩擦抵抗	D_θ	2.00e−7 $[Nms/rad]$
車軸の粘性摩擦抵抗	D_φ	1.00e−4 $[Nms/rad]$

(PUPPY取扱説明書より転写)

PUPPYのモデリング-1/7

ラグランジュ関数L と ラグランジュの運動方程式

L＝T－U

T ： 運動エネルギー
①車輪の回転運動エネルギー
②本体傾斜方向の回転運動エネルギー
③本体の並進運動エネルギー
④車輪の並進運動エネルギー

U ： ポテンシャルエネルー
①重心の位置エネルギー

PUPPYのモデリング-2/7

ラグランジュの運動方程式

$$\begin{cases} \dfrac{d}{dt}\left(\dfrac{\partial}{\partial \dot{\varphi}} L\right) - \dfrac{\partial}{\partial \varphi} L + D_\varphi \dot{\varphi} = \tau \\ \dfrac{d}{dt}\left(\dfrac{\partial}{\partial \dot{\theta}} L\right) - \dfrac{\partial}{\partial \theta} L + D_\theta \dot{\theta} = 0 \end{cases}$$

(PUPPY取扱説明書より転写)

PUPPYのモデリング-3/7

倒立状態では、 θ は小さいので、
cos θ ≈ 1 , sin θ ≈ θ と近似し、dθ/dt 等の非線形項をゼロとみなす

$$\begin{cases} a\ddot{\varphi} + (a+b)\ddot{\theta} + D_\varphi \dot{\varphi} = \tau \\ (a+b)\ddot{\varphi} + (a+2b+c)\ddot{\theta} - u\theta + D_\theta \dot{\theta} = 0 \end{cases}$$

(PUPPY取扱説明書より転写)

PUPPYのモデリング-4/7

状態方程式の形にする

$$\frac{d}{dt}\begin{bmatrix} \theta(t) \\ \theta(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \theta(t) \\ \dot{\theta}(t) \\ \dot{\varphi}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ b_2 \\ b_3 \end{bmatrix} \tau(t)$$

(PUPPY取扱説明書より転写)

これを、離散化する必要がある。

PUPPYのモデリング-5/7

周期Tで制御する離散時間制御モデルにする

$$\mathbf{x}(k+1) = e^{\mathbf{A}T}\mathbf{x}(k) + \int_{kT}^{(k+1)T} e^{\mathbf{A}((k+1)T-s)}$$

$$= e^{\mathbf{A}T}\mathbf{x}(k) + \int_0^T e^{\mathbf{A}s} ds \mathbf{B}\mathbf{u}(k)$$

(PUPPY取扱説明書より転写)

PUPPYのモデリング-6/7

また、

$$\mathbf{A}_D = e^{\mathbf{A}T} \quad \mathbf{B}_D = \int_0^T e^{\mathbf{A}s} ds \mathbf{B}$$

とおくと、

$$\mathbf{x}(k+1) = \mathbf{A}_D \mathbf{x}(k) + \mathbf{B}_D \mathbf{u}(k)$$

（PUPPY取扱説明書より転写）

PUPPYのモデリング-7/7

状態方程式の形の離散時間制御モデル

$$\begin{bmatrix} \theta(k+1) \\ \dot{\theta}(k+1) \\ \dot{\varphi}(k+1) \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \theta(k) \\ \dot{\theta}(k) \\ \dot{\varphi}(k) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} \tau(k)$$

（PUPPY取扱説明書より転写）

これが、制御設計の基本式になります

PUPPYの倒立

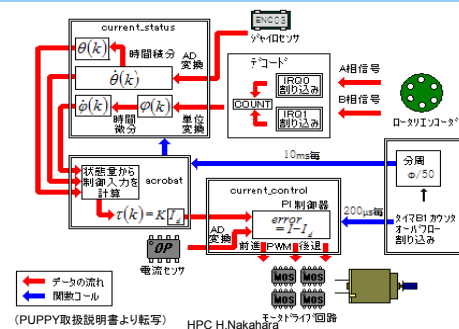
･･･一部省略･･･

位置は、制御しない。動きながら倒れない。

倒立（走行）制御の目標

- 本体の重心回りの角速度d θ /dt ＝0
- 車輪の回転角速度　d ϕ /dt : 目標値に追従させる

PI制御

既存のPUPPY制御プログラム

（PUPPY取扱説明書より転写）

XMOSのチップで実現

基本方針

- XS1-L1 64pin を使用（1コア、8スレッド）
- PUPPY本体の既存の信号を全て取り込む
- 機能は、すべてソフトウェアで実現･･･

ただし、追加したIC

- ジャイロ用ADコンバータ
- モータ電流値用ADコンバータ

XCore

・開発環境：Eclipse

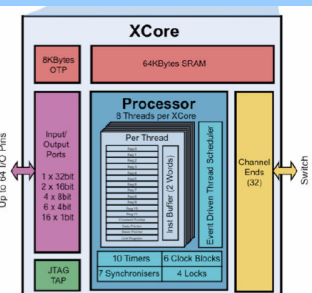
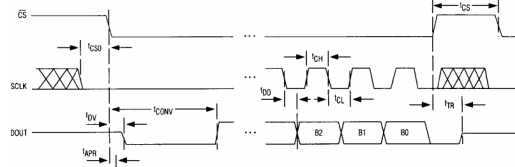
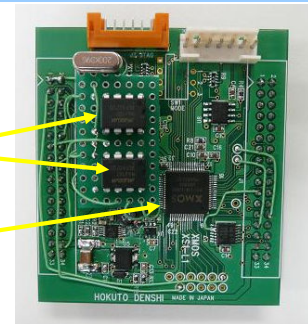
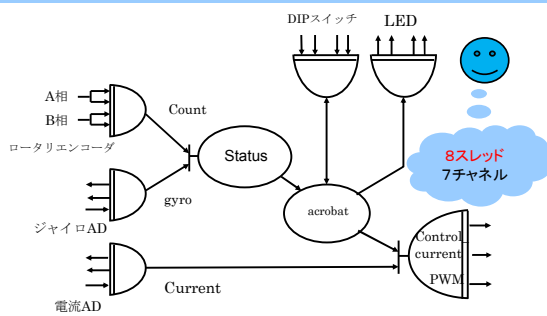
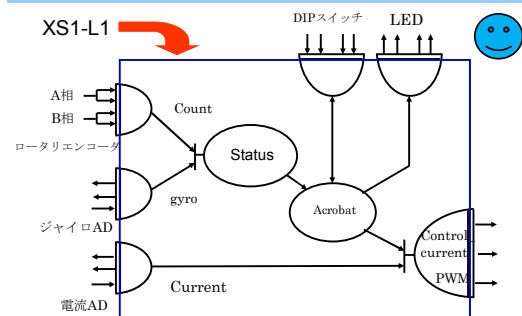
・開発言語：XC言語

・64KBのメモリ

・32個のChannel End

・64pinのI/O pin

・8Thread (ハードウェアのスレッド、処理装置)

XS1-L1は、このコア1個

ADコンバータ
MAXIM MAX187
3線シリアルインターフェース
逐次比較型
12ビット分解能
HPC H.Nakahara
19

試作ボード
ADC(MAX187)
XS1-L1
HPC H.Nakahara
20

ソフトウェアの構成
DIPスイッチ
LED
A相
B相
ロータリエンコーダ
Count
Status
gyro
ジャイロAD
acrobat
8スレッド
7チャネル
Control
current
PWM
Current
電流AD
HPC H.Nakahara
21

ソフトウェアの構成
XS1-L1
DIPスイッチ
LED
A相
B相
ロータリエンコーダ
Count
Status
gyro
ジャイロAD
Acrobat
Control
current
PWM
Current
電流AD
HPC H.Nakahara
22

メインプログラム
int main() { chan Gyro,Count,Current,DIPch,LEDch, Status,ID;
par{ Gyro_AD(Gyro_CS, Gyro_SCLK, Gyro_Data, Gyro);
RotaryENC(A_pos, A_neg , B_pos, B_neg, Count);
Current_AD(Cur_CS, Cur_SCLK, Cur_Data,Current);
Status_X(Count, Gyro,Status);
Acrobat_X(Status, ID,DIPch,LEDch ,type , time);
Control_Cur_X(ID,Current, PWMp);
DIPswitch(DIPch, DIPp);
LED4(LEDch, LEDp);
}
return 0;
}
DFDそのまま！
HPC H.Nakahara
23

難　題
難　題
・XCでは、浮動小数点型が使えない　！！
・したがって、チャネル通信もできない　！！
Status、Acrobatプロセスどうする？
プロセス間のデータ受け渡し可能か？
対　策
・Cファイル（拡張子．Cのファイル）に記述すると浮動小数点型が利用可。
・チャネル通信（但し整数）、xccompat.h　あり
・浮動小数点型を整数部と少数部の二つの整数で表現、この二つの整数をチャネル通信する。
HPC H.Nakahara
24

Cファイル（拡張子. C）内の対策

```
#include <xccompat.h>
#define MAXI 100000000.0f   // 小数部のためのscale定数
int  getIntP ( float x ) {
        int x_int ;
        x_int = ( int ) x ;
        return  x_int;
  }

int  getDecP ( float  x ) {
        int x_int ,  x_dec ;
        float  scale = MAXI ;
        x_int = ( int ) x ;
        x_dec = ( x - ( float ) x_int ) * scale ;
        return  x_dec ;
}
```

整数部の取得

少数部の取得

Cファイル（拡張子. C）内の対策

```
float  getFloat ( int  x_int , int  x_dec ) {
        float  x , y ;
        float  scale = MAXI ;
        x = ( float ) x_int ;
        y = x + ( ( float ) x_dec ) / scale ;
        return  y ;
}
```

浮動小数点型の取得

チャネル通信(整数のみ) 、プロトタイプの宣言

```
int   ReadCh ( chanend  ch ) ;        // xcファイルで定義
void  WriteCh( int  d , chanend ch); // xcファイルで定義
```


XCファイル内の対策

チャネル通信

・チャネル入力、ReadChの定義

```
int  ReadCh( chanend ch ) {
        int tmp ;
        ch :> tmp ;
        return  tmp ;
}
```

整数のチャネル入力、これをCファイルで使う

・チャネル出力、WriteChの定義

```
void  WriteCh ( int  d , chanend ch ) {
         ch <:d  ;
}
```

整数のチャネル出力、これをCファイルで使う

XCのメインプログラム

```
int  main()  {  chan   Gyro,Count,Current,DIPch,LEDch, Status,ID;
   par{   Gyro_AD( Gyro_CS, Gyro_SCLK, Gyro_Data, Gyro);
          RotaryENC( A_pos, A_neg , B_pos, B_neg, Count);
          Current_AD( Cur_CS, Cur_SCLK, Cur_Data,Current);
          Status_X(Count, Gyro,Status);
          Acrobat_X(Status, ID,DIPch,LEDch ,type , time);
          Control_Cur_X( ID,Current, PWMp);
           DIPswitch( DIPch, DIPp);
          LED4(LEDch, LEDp);
   }
return 0;
}
```

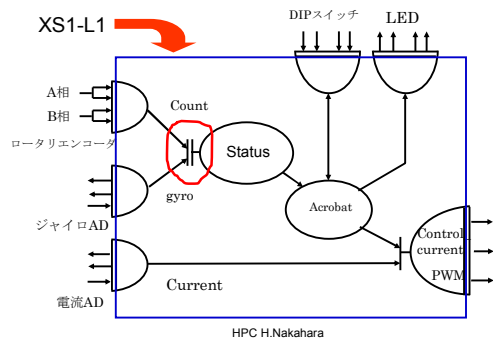
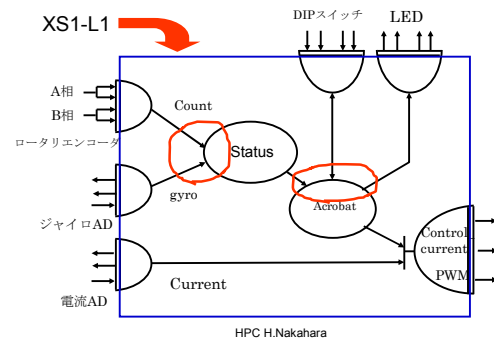
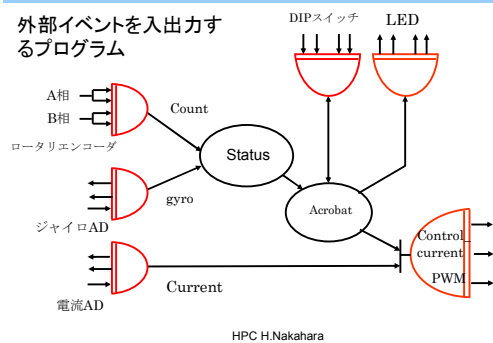
Cファイルを利用！！

これで良いか！？

```
int  main()  {  chan   Gyro,Count,Current,DIPch,LEDch, Status,ID;
   par{   Gyro_AD( Gyro_CS, Gyro_SCLK, Gyro_Data, Gyro);
          RotaryENC( A_pos, A_neg , B_pos, B_neg, Count);
          Current_AD( Cur_CS, Cur_SCLK, Cur_Data,Current);
          Status_X(Count, Gyro,Status);
          Acrobat_X(Status, ID,DIPch,LEDch ,type , time);
          Control_Cur_X( ID,Current, PWMp);
           DIPswitch( DIPch, DIPp);
          LED4(LEDch, LEDp);
   }
return 0;
}
```

DFDそのままだが･･･

Statusプロセスの例

```
void Status_X(chanend Count_ch , chanend Gyro_ch , chanend
                                                  status_ch )
{   int Count=0, Gyro=0;
    while(1){
      select{
        case Count_ch:> Count:
            break;
        case Gyro_ch:> Gyro :
               status_core(Count , Gyro , status_ch);
            break;     }
      }
}
```

ポート割り付け（1／3）

ポート割り付け

	1b	4b	8b	16b	XC-5	PUPPY
XD0	P1A0				SPI MISO	
XD1	P1B0				SPI SS	
XD2		P4A0	P8A0	P16A0	CLOCKLED 0	PWM、前、Forward
XD3		P4A1	P8A1	P16A1	CLOCKLED 0	PWM、後、Back
XD4		P4B0	P8A2	P16A2	CLOCKLED 1	DIP SW1/4
XD5		P4B1	P8A3	P16A3	CLOCKLED 1	DIP SW2/4
XD6		P4B2	P8A4	P16A4	CLOCKLED 1	DIP SW3/4
XD7		P4B3	P8A5	P16A5	CLOCKLED 1	DIP SW4/4
XD8		P4A2	P8A6	P16A6	CLOCKLED 0	PWM、信号
XD9		P4A3	P8A7	P16A7	CLOCKLED 0	
XD10	P1C0				SPI CLK	
XD11	P1D0				SPI MOSI	

HPC H.Nakahara 40

ポート割り付け（2／3）

	1b	4b	8b	16b	XC-5	PUPPY
XD12	P1E0				CLOCK R	ロータリ A相1/2、Positive
XD13	P1F0				CLOCK G	ロータリ、A相2/2、Negative
XD14		P4C0	P8B0	P16A8	CLOCKLED 2	
XD15		P4C1	P8B1	P16A9	CLOCKLED 2	
XD16		P4D0	P8B2	P16A10	BUTTON [A]	
XD17		P4D1	P8B3	P16A11	BUTTON [D]	
XD18		P4D2	P8B4	P16A12	BUTTON [C]	
XD19		P4D3	P8B5	P16A13	BUTTON [B]	
XD20		P4C2	P8B6	P16A14	CLOCKLED 2	
XD21		P4C3	P8B7	P16A15	CLOCKLED 2	
XD22	P1G0				PROTOTYPE	ロータリB相1/2、Positive
XD23	P1H0				UART TX	
XD24	P1I0				UART RX	
XD25	P1J0				PROTOTYPE	ジャイロADC1/3、CS

HPC H.Nakahara 41

ポート割り付け（3／3）

	1b	4b	XC-5	PUPPY
XD26		P4E0	BUTTONLED A	LED1/4
XD27		P4E1	BUTTONLED B	LED2/4
XD32		P4E2	BUTTONLED C	LED3/4
XD33		P4E3	BUTTONLED D	LED4/4
XD34	P1K0		SPEAKER	ロータリB相2/2、Negative
XD35	P1L0		PROTOTYPE	ジャイロADC2/3、SCLK
XD36	P1M0		PROTOTYPE	ジャイロADC3/3、DATA
XD37	P1N0		PROTOTYPE	電流ADC1/3、CS
XD38	P1O0		PROTOTYPE	電流ADC2/3、SCLK
XD39	P1P0		PROTOTYPE	電流ADC3/3、DATA

HPC H.Nakahara 42

ロータリエンコーダ

```
void RotaryENC( in port  A_pos , in port  A_neg  , in port  B_pos , in port
    B_neg , chanend  Count_Ch ) {   //  A_posとA_negには、同じ信号が入る
 int count=0  ,   unsigned HL;
  while(1){
   select {    case A_pos when  pinsneq(0x0) :> void : // A相の立ち上がり
               B_pos :> HL;
               if ( HL==0 ) { count=count+1; }
               else { count=count-1;}     break;
        case A_neg when pinsneq(0xf)  :> void : // A相の立ち下がり
                 B_pos :> HL;
                 if ( HL==1) {  count=count+1; }
                 else{ count=count-1;}         break;
       case B_pos when pinsneq(0x0) :> void :  //B相の立ち上がり
          ・・・・ 省略  ・・・・  }  }
  Count_Ch <: count;
  }
}
```


ジャイロADC

```
void Gyro_AD(out port CS,out port SCLK,in port Data,chanend Gyro){
   int d ,  offset ;
   waitms(500);  // 電源ONの0.5秒後に測定開始
   for (int i=0;i<100;i=i+1){    //100回、1秒間測定
      offset=offset+ADC12( CS, SCLK, Data);
           waitms(10);
   }
   offset=offset/100;     //    ====== offset の取得
   while(1){
     d=ADC12( CS, SCLK, Data) ;
     Gyro <:(d - offset)/4;     // 10  bit  ADCに換算  !!!
            waitms(10);      // 10ms このタイミングで各プロセスが同期する。
   }
}
```


電流ADC

```
void Current_AD(out port CS,out port SCLK,in port Data,chanend Cur_AD){
     unsigned tm;
     int d;
     timer t;
     t:> tm;          // 開始時刻の取得
     tm=tm+20000;     // 200micro 後を設定
     while(1){
           d=ADC12( CS, SCLK, Data)  ;
           Cur_AD <:d/4;        // 10bit ADCに換算
           t when timerafter ( tm )  :>  void ; //200microになるまで待つ
           tm=tm+20000;     // 200micro 後を設定
     }
}
```


ADコンバータ(ADC12)1/2

```
int ADC12(out port CS,out port SCLK,in port Data){
    unsigned now  , unsigned AD_Data=0 , tmp=0 , timer  t ;
    AD_Data=0;
    CS<:0x0;       // Start
    t:>now;
    t when timerafter ( now +1000) :> now ; // COnversion Time 10 micro
    for ( int  I = 0 ; i<12 ; i=i+1 ) {       // 12bit  取得
      SCLK<:0x01;
      t when timerafter ( now  + 50 )  :> now;  // 500 ns
      SCLK<:0x0;  //
      t when timerafter ( now +10) :> now;  // 100 ns
           Data :> tmp;
           AD_Data=AD_Data*2+tmp;  // 最初にMSBが来る,Shift plus 0bit目
           t when timerafter ( now +10) :> now;  // 100 ns
    }
    ・・・・    次へ  ・・・・        }
```


ADコンバータ(ADC12)2/2

```
int ADC12(out port CS,out port SCLK,in port Data){
    ・・・
    for(int i=0;i<12;i=i+1){      // 12bit  取得
      ・・・
    }
    SCLK<:0x01;   // 13パルス目
    t when timerafter ( now +50) :> now;  // 500 ns
    SCLK<:0x0;
    CS<:0x1;       // END
    t when timerafter ( now +50) :> now;  // 500 ns
   return (int)AD_Data;
}
```


DIPスイッチ

```
void DIPswitch ( chanend DIPch , in port DIPsw ){
    //  DIPsw は、4bit Port
    //  DIPch は、双方向
    int st = 0;
    DIPch :> int _;
    DIPsw :> st;
    DIPch <: st;

}
```

LED4

```
void LED4 ( chanend inch , out port LED){
    //  LEDは、4 bit Port
    int LED_st = 0;
    inch  :>  LED_st;
    LED <:  LED_st;

 }
```


Control_Cerrent 1/2

```
void Control_Cur_X ( chanend ID_ch , chanend Cur_AD , out port  PWMp ) {
    int  CurrentAD , Id_Union , PWM_u , Pw_unit=200;   //  200 microの1%
     unsigned Pwdt, PWMdata=0;

     while(1){
     select{
           case ID_ch : >Id_Union:   //  電  流  の目標値、整数
                 break;
           case Cur_AD  :> CurrentAD:  // 200 micro毎、現在の電  流
              ････  次のページ、PWM  出力  ････
              break;
           }
     }
}
```

現在の電 流 と目標値からPWMを出力

Control_Cerrent 2/2

```
PWM_u= control_crrent_core(Id_Union, CurrentAD);
 if ( PWM_u > 0) {    PWMdata=0x01 ; }
  else if ( PWM_u < 0){  PWMdata=0x02;
                         PWM_u = -PWM_u; }
  else {  PWMdata=0x03;
             PWM_u     = MIN_DUTY ; }
 if ( PWM_u > MAX_DUTY )  PWM_u = MAX_DUTY; // 最大値を設定
 if ( PWM_u < MIN_DUTY )  PWM_u = MIN_DUTY ; // 最小値を設定

// ここでPWM信号を発生、周期  200micro秒
   Pwdt = Pw_unit*PWM_u ;  // パルス幅の計算
   PWMdata = PWMdata+0x04;  // 3bit目を1にする。
   PWMp <: PWMdata;
    waitmicro( Pwdt );
   PWMp<:0x0;
```


Control_Cerrent_Core

```
int control_crrent_core(int Id_Union,int Cur_AD)   {
     union U_FloatInt  U_tmp;
     float       u;                              // 出力するPWM算出用
     float       err;                            // 制御偏差
     static float    sum = 0.0f;                 // 制御偏差の積算値
      float Id;
      U_tmp.Int = Id_Union;
       Id = U_tmp.Float;                         // Id のfloatを得る
       err = Id - ( I_C * ( float ) Cur_AD ) ;   // 電  流   の目標値と現在の偏差
       u    = KP_MOT * err + KI_MOT * sum ;    / / 制御入力を計算
       sum +=   err;                                    // 積算誤差を変数に格納
       return  ( int ) u ;
 }
```


status_core

```
void  status_core ( int  count , int  gyro , chanend  status_ch)
 {     float dTheta   , ThetaG   ,Fai;
        static int     count_1 = 0;        // 1時刻前のカウントを格納
        static float    dtheta_1 = 0;     // 1時刻前の車体角速度を格納
       dTheta = G_C * ( float ) gyro ;  // 車体の角速度を算出
      ThetaG += T_T / 2.0f * ( dtheta_1 + dTheta)  ; // 角速度を積分
       dFai = ( float ) ( count  - count_1) * K_C / T_T + dTheta ; //角速度
       count_1 =count;                          // エンコーダカウント値を保存
       dtheta_1=dTheta;                         // 車体角速度の値を保存

      // ここから、チャネルへ出力
        WriteFloat ( dTheta , status_ch) ;
        WriteFloat ( ThetaG , status_ch) ;
        WriteFloat ( dFai , status_ch) ;
  }
```


終わりに

CSPモデルプログラム全般に関して

http://www.cspjapan.org/

http://www.cspjapan.org/JCSP/jcsp.htm

http://www.cspjapan.org/JIBUNET/jibunet.htm